Machine Learning in Implementing Policies for Squid Proxy Server

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Sudhakar Aluri

to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

March, 2003

Certificate

This is to certify that the work contained in the thesis entitled " *Machine Learning in implementing policies for Squid Proxy Server* ", by *Sudhakar Aluri*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

February, 2003

(Dr. Harish Karnick)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

Today the Internet has become an integral part of every ones life, as many services like mail, news, chat are available and huge amounts of information on almost any subject is available. However, in most cases the bandwidth to connect to the internet is limited and especially in educational institutions and companies one would like to implement policies such that productive usage is given preference in bandwidth allocation.

Squid proxy server is a full-featured web proxy, which increases the efficiency of the Internet link by providing caching and proxy services. Squid provides many mechanisms to set access control policies. However, deciding which policies to implement requires experimentation and usage statistics that must be processed to obtain useful data. We propose an Architecture based on machine learning to determine policies depending on the content of current URLs being visited. The main component in this architecture is the Squid traffic Analyzer, which classifies the traffic and generates URL lists. These URL lists are used in formulating access policies.

We introduce the concept of delay priority, which gives more options to system Administrators in setting policies for bandwidth management. As Squid allows HTTP tunneling, it forms a loophole for strict policy management. We also study proxy tunneling in Squid and suggest some possible solutions to this problem.

Acknowledgements

I take this immense opportunity to express my sincere gratitude toward my supervisor Dr. Harish Karnick for his invaluable guidance. It would have never been possible for me to take this work to completion without his relentless support and encouragement. I consider myself extremely fortunate to have had a chance to work under his supervision. Inspite of his hectic schedule he was always approachable and took his time off to attend to my problems and give the appropriate advice. It has been a very enlightening and enjoyable experience to work under him.

I also wish to thank whole heartily all the faculty members of the Department of Computer Science and Engineering for the invaluable knowledge they have imparted to me and for teaching the principles in most exciting and enjoyable way. I also extend my thanks to the technical staff of the department for maintaining an excellent working facility. I would like thank Mr.Navpreet singh and CC staff for helping in my experimental setup.

My stay at IITK was unforgettable to say the least, and the biggest reason for it being my classmates of the great mtech2001 batch. I thank my classmates just for being such great buddies.

I would like to thank Atul kumar, amber and pathy for their prompt help, whenever I need assistance.

I would like to thank my parents for taking me to this stage in life, it was their blessings which always gave me courage to face all challenges and made my path easier. I would like thank my brothers who helped me through out my education.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Bandwidth is an important resource in the Internet. It needs to be used efficiently and more importantly productively. Generally, bandwidth is distributed among groups of users based on some policy constraints. However, it turns out that the users do not always use the entire allocated bandwidth at all times. Also, some times they need more bandwidth than what is allocated to them. Ideally, *productive* usage should be preferred over *unproductive* usage when bandwidth is scarce. But when it is abundant then any kind of use can be permitted provided it is in consonance with policy. The bandwidth usage patterns of users vary with time of the day, time of the year and requirements. So there is a need for *dynamic allocation of bandwidth* that satisfies the requirements of the users, manages variable usage and is consistent with administrative usage policy.

Internet usage is varied and in the context of an institution or organization an administrator would like to maximize *productive* usage. There is, therefore, a need to implement control access policies, which prevents unproductive use but at the same time does not, to the extent possible, impose censorship .

Squid is a full featured web proxy cache [4]. It supports proxy services and caching of protocols like HTTP, FTP etc. It also supports many Internet caching protocols

like ICP, HTCP etc. The performance of squid proxy depends on many configuration parameters (policies) of bandwidth, caching and access controls.

Squid proxy is generally employed at the end of a local network and the starting point of an Internet link. Typically the allowed bandwidth on Internet links is limited, ranging from kbps to Mbps. So there is a need for better utilization of this resource and for finding better policies.

In the context of limited bandwidth an organization would like to use it for productive purposes like data of academic relevance in an academic institution. However, Internet usage ranges from research, news, entertainment, sports, general information to downloading movies, music and pornography. So there is a need for traffic analysis to find the shape of the traffic in terms of content and help system administrators in setting policies for shaping the traffic. Traffic analysis can categorize URLs and IP Addresses into categories like academic, sports, music etc. This categorization can be used to allocate bandwidth according to policies adopted by the organization.

Squid has a set of mechanisms for implementing policies based on many configuration parameters like source, destination, domains, time, regular expression on URLs etc. Squid allows HTTP tunneling to support SSL and TLS. But, these provide a loophole for overcoming policies. Generally, users use mass-downloaders or some other tools which use HTTP tunneling [6] [3] to over come access control policies. So there is a need to find a way for enforcing policy strictly.

1.2 Problem Statement and Contribution of this Thesis

Problem Statement: *Discovering and implementing environment and usage sensitive policies in Squid proxy server that lead to better utilization of bandwidth and other resources.* We have adopted machine learning techniques to study the

current environment and usage patterns. System administrators can then use this information to decide and implement policy by making use of the existing tools available in Squid to control access.

This thesis makes the following contributions to the problem:

- Priority over *delay pools* [5]: To give more options to the system administrator to define bandwidth limits and distribute unused bandwidth to chosen user groups we have introduced the concept of *delay priority* over *delay pools*. This extends the existing *delay pools* mechanism by adding priority to it.

- Machine learning approach for categorization: A naive Bayesian classifier has been used to classify URLs based on the URL itself and the content of URL visited.

- Possible solutions to HTTP tunneling: We have studied the concept of HTTP tunneling and propose possible options to enforce policy more strictly.

- An interface for the system administrator, has been built to allow him/her to configure policy parameters.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we discuss the various policy mechanisms of Squid and Squid log analyzers. Chapter 3 discusses the design of delay priority and the architecture of the Squid traffic classifier. In chapter 4, we discuss the implementation details of each component. In Chapter 5, we present a study on HTTP tunneling. In chapter 6, we discuss experiments done and the results of our work. In Chapter 7, we present the conclusions and briefly discuss how the work can be extended.

Chapter 2

Relative Work

In this chapter we briefly discuss the Squid proxy server and review previous work on bandwidth allocation and policy enforcement.

2.1 About Squid Proxy Server

Squid is a full-featured web proxy caching server. It supports protocols like HTTP, FTP, SSL, GOPHER, NEWS etc. It also provides Internet caching protocols like ICP, HTCP, WCCP, CARP and Cache Digests. It provides transparent caching and reverse proxying. To enhance performance it also provides the caching of DNS lookups. To communicate with other networking tools it supports the SNMP protocol. It provides a flexible access control mechanism to set policy. It provides many configuration parameters to tune it to a particular environment. The configuration parameters range from simple port numbers on which it provides HTTP services to cache file system parameters, cache replacement algorithms and other more complex configuration hooks. Squid performance can be substantially altered by tuning configuration parameters and using the supported mechanisms to implement policy [4][17].

2.2 Bandwidth Management

Squid provides a concept called *delay pools* [5], to manage bandwidth. In many countries insufficient bandwidth results in slow access to the Internet. This situation is exploited by a small number of people who use a disproportionate share of available bandwidth for services like file downloads etc, starving other users. Therefore some means of controlling and managing bandwidth is necessary. Many Organizations, ISPs provide Internet access through some proxy-cache servers to reduce load on the Internet link. Squid is one of the more popular proxy cache servers, which provides bandwidth management.

The bandwidth management component in Squid is called *delay pools* and was written by David Luyer [8]. *Delay pools* allows management of bandwidth by defining different *classes* of users and *max*, *restore* parameters. The idea came from University of Western Australia, where they wanted to restrict student traffic (without affecting staff traffic, while maintaining the same level of local peering and cache hits).

Here we define some technical terms before discussing the concept of *delay pools*.

- bucket

  an individual delay bucket represents a traffic allocation which is replenished at a given rate (up to a given limit) and causes traffic to be delayed when empty.

- bucket group

  a group of buckets (within a pool), such as the per-host bucket group, the per-network bucket group or the aggregate bucket group (the aggregate bucket group is actually a single bucket).

- pool

  a collection of bucket groups as appropriate to a given class.

- class

  the class of a delay pool determines how the delay is applied, i.e., whether the different client IPs are treated separately or as a group (or both).

- class 1

  a class 1 delay pool contains a single unified bucket (aggregate) which is used for all requests from hosts within the pool.

- class 2

  a class 2 delay pool contains one unified bucket (aggregate) and 255 buckets, one for each host on an 8-bit network (IPv4 class C).

- class 3

  contains one unified bucket (aggregate), 255 buckets for the subnets in a 16-bit network, and individual buckets for every host on these networks (IPv4 class B)

The *delay pools* concept uses the *Token bucket Algorithm* [2] for managing the bandwidth.

Each bucket has two parameters *max* and *restore*. The *max* value defines the size of the token bucket and *restore* defines the rate at which tokens are put in to the bucket. Here we are explaining the working of delay pool of class 1. The class 1 pool has only one bucket with *max, restore* rates specified.

Let the current capacity of the bucket be x_b kbits. For each one second, tokens of size restore(x_r kbits) rate are put into the bucket. This means x_b is increased with restore rate, x_r kbits per second. If x_b crosses the *max* value of the bucket, x_b is set to *max* capacity of the bucket and the excess tokens are lost.

If a user of this pool requests x_1 kbits, then the request is serviced as follows: if the bucket contains x_b kbits as accumulated by the tokens put into the bucket and $x_b \geq x_1$ then the user request for x_1 kbits is allowed and the x_b is reduced by x_1 kbits. If $x_b < x_1$ then the user request is allowed for only x_b kbits and x_b is set to 0 kbits.

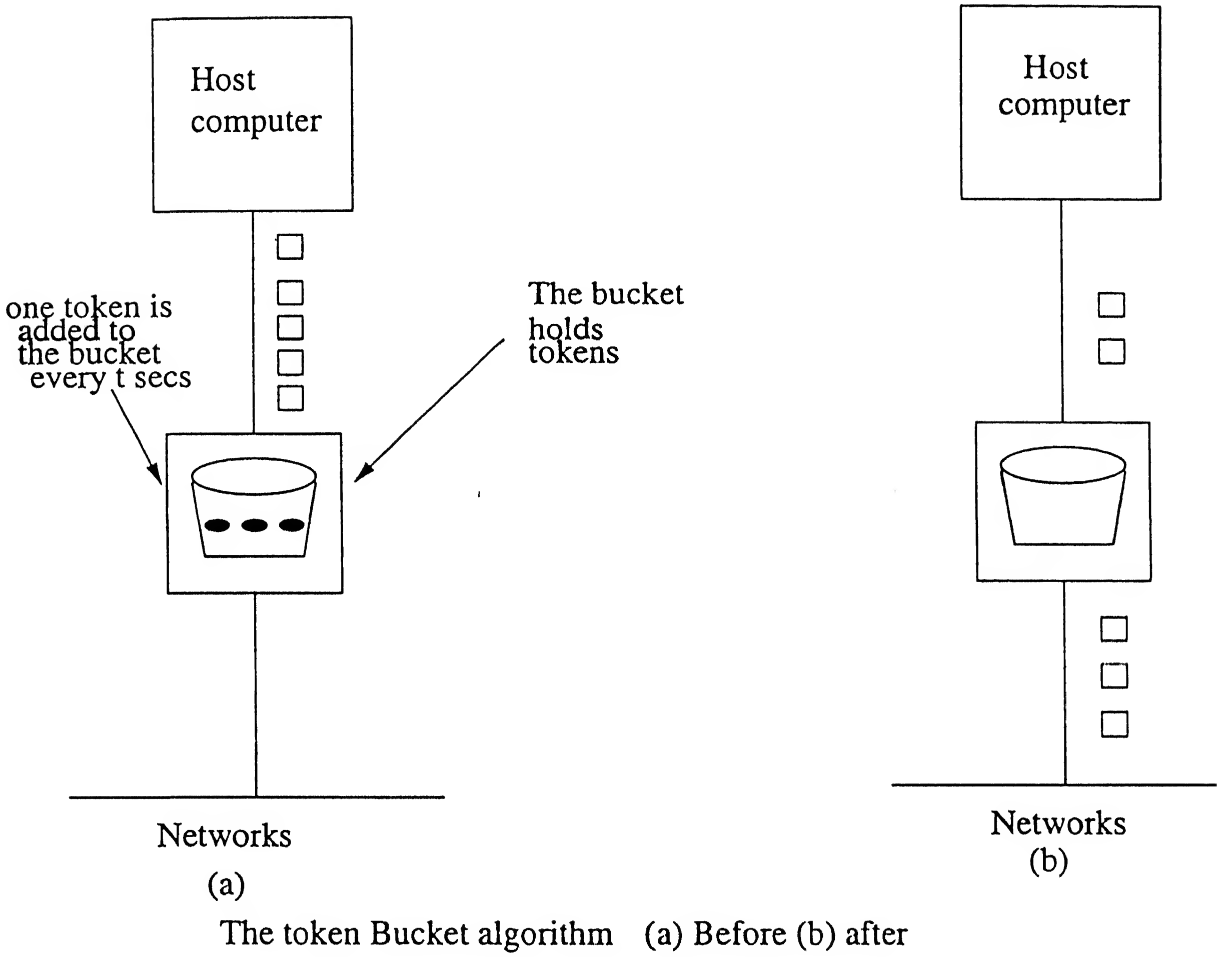

Figure 2.1: Token Bucket Algorithm

When this feature is used time-averaged bandwidth available to each user is limited by the rate at which tokens are issued into the bucket. However if a user has not requested data for some time, his tokens are accumulated (at most *max*), and the requests are serviced in a burst of accumulated-tokens-in-bucket speed. In this way an interactive user waiting for a page without opening many browsers simultaneously will get a reasonable response, while users who use many connections are limited by the restore rate. These limitations can also be set at aggregate level to control a group of users.

But there is a limitation in using *delay pools*. The *delay pool* parameters are statistically defined in the Squid configuration file and cannot be changed dynamically to suit the changing traffic conditions. Bandwidth usage patterns change considerably during the course of a day. An individual restore value that gives fair access to all users when traffic is high, is unduly restrictive during slack times. i.e. the unused bandwidth by certain users is wasted and will not be given to the other users accessing the link at that time. The users during the low traffic period are also restricted by the same restore rate. So a lot of bandwidth goes unused. One solution to this is to have dynamic configurations that depend on the traffic and reconfigure Squid by sending it the reconfigure signal. For example,

delay_parameters 2 64000/64000 16000/16000 600/8000 The policy (rule) says that delay pool number 2 is a class 3 delay pool, and we want to limit it to a total of 512kbps (strict limit) with each 8-bit network permitted 128kbps (strict limit) and each individual host permitted 4800bps with a bucket maximum size of 64kb. If all the hosts (users) are not using the link, then 4800bps bandwidth allocated for each host is wasted and not allowed to used by the other hosts. Each 8-bit network permitted 128kbps. If only one host is accessing the link in that 8-bit network, then 64kbps bandwidth is wasted. If only hosts in one 8-bit network are accessing the link, then 512-128 = 384 kbps is wasted.

The *delay pools* provide a set of parameters [6] that can be configured to manage bandwidth.

2.3 Policies in Squid

The Squid proxy server allows the administrator to set different policies to tune it to the environment.

2.3.1 Access Controls

Squid's access control scheme is relatively comprehensive and difficult to understand. Squid provides access control lists to control user access (i.e. allow/deny)

based on special criteria. Access Controls contain two concepts: *ACL elements and access lists*. An ACL element helps in defining the `criteria`. In all the examples of access control rules, we represented all the Squid keywords in bold, where as userdefined names in italic. For example,

acl *iitk_hosts* **src** *172.31.0.0/255.255.128.0 172.30.0.0/255.255.0.0 172.32.0.0/255.255.0.0*

The named ACL element *iitk_hosts* represents all the host IP addresses in IITK.

acl *bad_dest* **dst** *172.31.0.0/255.255.128.0*

The named ACL element *bad_dest* defines the IP addresses of form 172.31.*.* with netmask 255.255.128.0.

acl *cc_users* **proxy_auth REQUIRED**

The named ACL element *cc_users* represents the users who get authenticated.

Access lists help to allow/deny the traffic satisfying the criteria as defined in acl elements. For example the rule,

http_access allow *iitk_users cc_users*

allows the traffic/requests, if the requests are from IITK machines and get authenticated.

http_access deny *bad_dest* # this access rule denies the traffic that matches one of the destination addresses represented by *bad_dest*.

2.3.2 Access Elements

Squid predefines certain types of ACL elements. Using these types an Administrator can define the named ACL elements. In the Above example *src, dst, proxy_auth* are ACL types, where as *iitk_hosts, bad_dest, cc_users* are named ACL elements. An ACL type can be *src (source), DST (destination), srcdomain (source domain), dstdomain (destination domain), srcdomain_regex, dstdomain_regex (regular expression on source, destination domains), time (time of the day), url_regex, urlpath_regex (regular expressions on URL or paths), port (port number), proto (protocols), method (HTTP methods), browser, ident (user name)*, etc.

For example,

acl *SSL_ports* **port** *443 513*
the named ACL element *SSL_ports* defines the ports 443, 513.
acl *mymethod* **method CONNECT**
the named ACL element *mymethod* defines the method CONNECT.

Each ACL element is assigned a unique name. A named element consists of a list of values. When checking for a match the multiple values use *OR* logic. In other words, an ACL element is matched when any one of its values is a match. We cannot give the same name to different ACL elements. It will generate a syntax error. We can put different values for the same ACL name on different lines. Squid combines it into one list.

Some ACL elements cause processing delays. For example use of *src_domain, src_domain_regex* require a reverse DNS lookup on client's IP address. This lookup adds some delay to the request.

To activate some of the ACL elements (e.g. *arp, snmp_community*) compile time switches are needed, for example *-enable-arp-acl, -enable-snmp* configuration option.

2.3.3 Access Lists

Squid provides a set of *access lists* to control traffic and to enhance performance. An *access list* rule contains the *allow* or *deny* keyword followed by a list of named ACL elements. Access list rules are checked in the order they are written, list searching is stopped, if a match is found. If a rule uses multiple ACL elements it uses *AND* logic. If none of the rules match, then the default action is the opposite of the last rule in the list. For example,
http_access deny *mymethod !SSL_ports*
http_access allow *cc_users iitk_hosts*
http_access deny *porn_sites* # assume *porn_sites* named ACL element is defined.
http_access deny *all* # assume the ACL named element *all* is defined as all the hosts.

These set of rules denies requests with the CONNECT method on other than SSL_ports and that match the porn_sites. It allows the traffic/requests that come from IITK hosts and which get authenticated. If first three rules do not match, then the last rule makes that request to be denied as *all* matches any request.

There are lots of access lists, which define the behaviour of the cache, redirector, authenticator, and snmp modules. For example,

- acl QUERY urlpath_regex cgi-bin ? # All the URL paths that contain the cgi-bin word.

- no_cache deny QUERY # do not cache the responses of cgi-bin queries.

The access list no_cache can be used to control the behavior of cache as explained in the above example.

2.3.4 Caching

Squid allows the administrator to set cache parameters to improve the cache *hit* rate. The set of parameters includes the cache file system type, size, its hierarchy structure and location. It also allows the specification of several other parameters like the type of objects which are to be cached, objects not to be cached, cache replacement algorithm, etc. The other parameters which influence caching are selection of neighbour cache peer, cache digest [15] collection etc. The selection of neighbour tells Squid the cache server to which a request has to be forwarded. A cache digest is a summary of contents of an Internet Object caching server. By exchanging cache digests between cache servers, requests can be serviced from the cache servers themselves. The presence of an object can be checked using the cache digests.

2.3.5 External Processes

Other than bandwidth management, caching and access controls there are several external processes that can be configured. The redirector process, ident lookup server, pinger, DNS child process etc.

- Redirector: Squid has the ability to rewrite the requests from clients. After checking the access controls, but before checking for cache hits, requested URLs may optionally be written to an external *redirector* process. This program, which can be highly customized, may return a new URL to replace the original request. Common applications for this feature are extended access controls and local mirroring.

- dnsserver: The standard system call *gethostbyname*(3) is a blocking system call, so an external process like dnsserver is used. It spawns many child processes to handle DNS requests.

- pinger: To get an accurate measure of RTT (round trip time) and to run a simple process like pinger as root rather than Squid proxy.

2.4 Traffic Analysis

Squid allows the system administrator to log traffic, so that it can be analyzed later. It stores different kinds of log files mainly access.log, store.log, cache.log, etc. The logs are a valuable source of information about Squid workloads. The log files record not only access information, but also system configuration errors and resource consumption (e.g. memory, disk space). Some of the log files have to be activated during compile time, others can be easily deactivated during run-time. To activate useragent.log, we have to enable at compilation time the option *–enable-useragent-log.* To activate other log files we have to configure in squid.conf file. For example, to activate the access.log,

cache_access_log /usr/local/squid/logs/access.log # the complete path represents the location of access.log file.

To disable the access.log file,

cache_access_log none. # this rule disables the access.log file in Squid proxy.

2.4.1 Cache.log

The *cache.log* file contains the debug and error messages. For log file analysis it does not have much to offer. This is meant for Squid developers who want to test new changes in Squid code.

2.4.2 Useragent.log

It gives details of browser distribution. This log should not be activated unless browser distribution is specifically needed.

2.4.3 Store.log

It logs the activities of the storage manager. It shows which objects are ejected from the cache, and which objects are saved and for how long. As a kind of transaction log it is usually used for analysis and debugging. A definitive statement, on whether an object resides on disk is only possible after analyzing the complete log file. The release (deletion) of an object may be logged at a later time than the swap out (save to disk).

The *store.log* file can provide data like: how many times a hot object was accessed, how much time a hot object spent in the cache (disk). With the knowledge of cache directory location, this log file allows for a URL to filename mapping without going through the cache disks.

The Contents in the store.log file are as follows.

- timestamp: when the line was logged in UTC with a milli second fraction.
- action: tells the location of the object and the action taken for that object.
  - SWAPOUT The object was saved to disk.
  - RELEASE The object was removed from cache (disk).
  - SWAPIN The object existed on cache (disk) and was read into memory.

- dirn : The directory number in the cache file system.
- filenumber: The file number for the object storage file in the cache file system.
- hashkey: It is the hash key of storage location.
- status: HTTP Reply status codes [16] like 200 (for OK), 404 (Not found).
- date: The value of HTTP Reply "date" header.
- lastmodified: The value of HTTP Reply "last modified" header.
- expires: The value of HTTP Reply expires header value
- content type: The HTTP Reply content type like text/html, image/gif etc.
- sizes: This field consists of two values separated by slash: 1. The advertised content length from HTTP "Content-Length:" reply header. 2. The size actually read. If the advertised length is missing, it will be set to zero. If the advertised length is not zero, but not equal to real length, the object will be released from the cache.
- method: The request method for the object e.g., *GET*
- key: The key to the object, usually URL.

For the sake of completeness, we give some entries in the store.log file.

1045109695.998 RELEASE -1 FFFFFFFF 9FAB787B518F5F69DEBCE33DC658 ABAA 407 -1 -1 -1 unknown -1/1322 GET http://www.tradeexit.com/link1.html.

The above record tells that the entry is made in to log file at 1045109695.998 UTC milli-sec and the object www.tradeexit.com/link1.html is released from the cache whose HTTP status code is 407. The -1 and unknown says that the other fields are undefined.

1045109696.026 SWAPOUT 00 00008AA7 E89D834C0C5283D9F63E64A68CD 1DDD 200 1045110929 1003266396 -1 text/html 2213/2213 GET http://me.engin.

umich.edu/newsandinformation/publications/annualreport/1995-1996/ar95-96/ResearchChart/resintchart.html

The above record tells that the entry is made in to logfile at 1045109696.026 UTC milli-sec and the object (as specified in the url) is cached in to cache file system in the directory 00, sub directory 08 and file name 8AA7. HTTP status code for these for these object is 200. Its content type is text or html. Its size is 2213. The last modified time for this object is 1003266396 UTC milli-sec, where as expire time is 1045110929. The hashkey E89D834C0C5283D9F63E64A68CD1DDD is used by Squid code.

1045109696.075 SWAPOUT 00 00008AA8 680FEBC4238FB0FC2DEAF4F6D055 6716 200 1045110766 1043022632 -1 image/gif 830/830 GET http://www.edevice.com/images/index/china.gif

1045109696.116 SWAPOUT 00 00008AAA E0D0605D3E5DA3F701DFE2A1821F BC80 200 1045110929 1026972217 -1 image/gif 43/43 GET http://www.nasscom.org/images/1px.gif

2.4.4 Access.log

Most log file analyzers are based on access.log. Currently it supports two file formats, one is the naive format, the other is the common log file format as defined by the CERN web daemon. The typical access.log record in the naive format contains the following entities: time, duration, client address, HTTP result codes, number of bytes transferred, HTTP request method, URL, identification, information regarding other cache peers, if they were involved in servicing the request, the content-type of the reply. There are many scripts and tools that can analyze these logs. Here we briefly discuss a few [12] .

- Calamaris [13]: Calamaris parses logfiles of Squid, other web proxy servers and generates reports in ascii/html format. The report contains a summary

of records parsed, time taken, in_traffic and out_traffic protocol wise. It also generates reports based on HTTP request types, response content-types(.gif, .jpeg, .css, .xml etc), Incoming TCP/UDP requests by HTTP status (HIT, MISS etc). It also gives the average and peak traffic.

- Webalizer [14]: Webalizer is a fast, free web server log file analysis program. It produces highly detailed, easily configurable reports in HTML format.

It generates reports that include the number of files, hits, misses, top sites, top URLs, top entry pages, top exit pages, top search strings and traffic load from top clients(who use lot of bandwidth). More usefully, they are represented graphically. There are other log analyzers[6] that provide filter options and generate different reports based on these filters.

2.5 Http Tunnelling

Squid supports encrypted protocols (like SSL/HTTPS/TLS) by tunnelling traffic between clients and servers. Squid can relay the encrypted bits between a client and a server.

Normally when a browser comes across the HTTPS URL, it does one of two things: either the browser opens an SSL connection directly to the origin server or the browser tunnels the request through Squid with the CONNECT request method. The HTTP CONNECT method is a way to tunnel any kind of connection through an HTTP proxy. The proxy doesn't understand or interpret the contents. It just passes bytes back and forth between the client and server. Squid does not (yet) encrypt or decrypt connections. But some patches that use OpenSSL allow Squid to do encryption/decryption.

Motivation

Squid offers several mechanisms for setting policies. The system Administrator has to decide the policies and implement them in terms of mechanisms provided

by Squid by setting configuration parameters in squid.conf. Once policies are set, Squid follows those policies. But the usage pattern of users varies and usually requires changes in policy.

The Squid policy mechanism is static and the System Administrator has to explicitly change policies to suit change in usage patterns. To set policies that suit the current environment, one has to understand and learn about the current environment and use that knowledge in deciding policies. We propose a technique based on *machine learning* that learns from the Squid environment and helps in arriving at policies that adapt to usage patterns.

Squid proxy does not have a priority concept. This can help in making policies dynamic and usage dependent - for example when unused bandwidth is available assign high bandwidth to a group of users. We have introduced a concept called *delay priority,* which will help in making policies dynamic and sensitive to usage.

Chapter 3

Present Work

3.1 Setup of Squid

IITK has 4 Internet Links with the following bandwidth:

- Old VSNL : 2 Mbits/s
- VSNL : 2 Mbits/s
- ERNET : 2 Mbits/s
- MLA : 512 Kbits/s

The Ernet link is used only sparingly in IITK and its bandwidth is available for outside colleges in the neighbouring region. As part of this work we have installed a Squid proxy server on the ERNET link and made it available to the IITK users. Though, it is a 2Mbps link, it effectively gives a maximum bandwidth of 1Mbps. The configuration of the proxy server was PII 133MHZ, 8GB hard disk and 32 MB RAM running Linux Redhat8.0 and Squid version Squid.2.4.STABLE7.

3.1.1 Initial Configuration

The Squid proxy was configured to allow all IITK users who had Computer Center logins. The authentication of users was done with username and password. No other configuration parameters to restrict the sites were set.

3.2 System Design

From an abstract point of view the problem can be seen as improving the performance of a proxy server (Squid) in a given environment. For the measure of performance we use two parameters a) better overall utilization of bandwidth within the constraints set for each delay pool and b) traffic control based on content (to model policy decisions).

We treat this problem in two parts. To improve bandwidth utilization we introduce the concept of priority in delay pools and to control traffic based on content we formulate the problem as a machine learning task and learn the appropriate class for a URL from given set of classes. We have used the log data and cache documents for training purposes in the learning task (classifying the documents). The learning itself is done by using a naive Bayes Classifier on the collected data. Once classification is done, policies based on these categories are used to tune the Squid proxy server. This process is continuous and can be applied till performance is improved to the required levels.

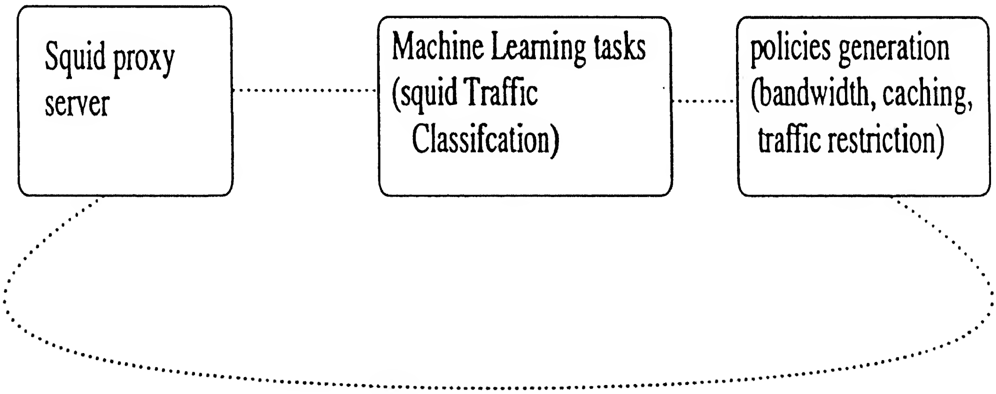

Figure 3.1: Architecture

The complete classification of Squid traffic is done in three different phase: data collection, learning and classification. In the data collection phase we have collected log data and documents from Squid proxy which characterize the traffic. The learning phase includes some user/manual interaction. It includes two steps namely *data*

cleaning and learning.

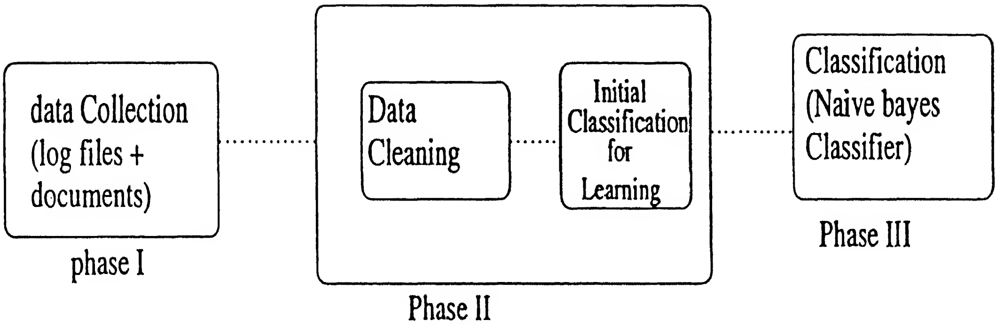

Figure 3.2: Phases in Squid Traffic Classification

3.3 Data Collection

We have collected the log files of Squid and documents from the cache file system. The collection of documents was done with an external process. We configured a Squid proxy server and enabled store.log, access.log and cache.log. As the sizes of the log files are in the range of 250MB - 500 MB, we rotated the log each day. Squid renames the old log files with integer extensions. We have zipped the log files to solve the problem of space on the system.

When users access Internet pages/data, Squid caches it based on the caching strategy and lifetime of the pages/data. Squid stores such cached pages in the *cachefs* and writes a log entry in *store.log*.

We have designed an external process ("Data Collector") which uses the store.log and *cachefs* to collect the documents. The store.log record gives information about the "location" of the pages/data in the *cachefs*. The external process, Data Collector copies the pages/data to the local directory or disk space.

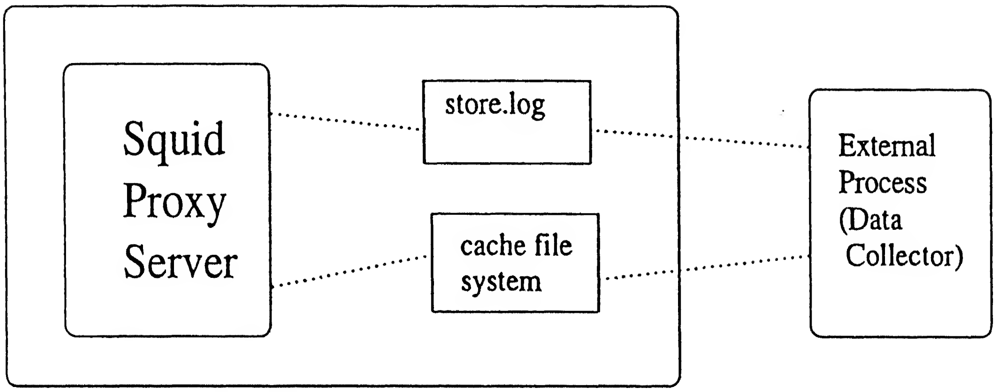

Figure 3.3: Data Collector

3.4 Classification of Traffic

The traffic through Squid proxy can be classified into different categories: news, sports, entertainment, music, science and technology, chat, movies, academic and pornography. The traffic is characterized by the log data and documents collected through the external process - Data Collector. We use the naive Bayesian Classification algorithm on these documents and classify them. The category of a document is identified using the content in the document or page. For example, if a document contains research/educational information it is considered to be in the academic category.

3.4.1 Bayesian Classification Algorithm

One highly practical Bayesian learning method is the naive Bayes learner, often called naive Bayes classifier. In document classification the algorithm performs better than almost all other competing algorithms [1].

The naive Bayes classifier can be applied to learning tasks where each instance x is described by a conjunction of attribute values and where the target function $f(x)$ can take on any value from some finite set V. A set of training examples of the target function is provided by the tuple of attribute values $< a_1, a_2, ..., a_n >$. In this case we wish to find the most probable category for an unknown URL (document)

also called the *maximum a posteriori* hypothesis, which we represent by v_{MAP}. The Bayesian approach to classifying a new instance is to assign the most probable target value, v_{MAP}, given the attribute values $< a_1, a_2, .., a_n >$ that describe the instance.

Bayes theorem is the basis of Bayesian learning methods because it provides a way to calculate the posterior probability $P(h|D)$, from the prior probability $P(h)$, together with $P(D)$ and $P(D|h)$.
Here $P(h|D)$ is the probability h holds given training data D (similarly, $P(D|h)$). $P(h)$ is the *a priori* probability that h holds and $P(D)$ the *a priori* probability that data D will be observed. Bayes theorem gives a formula to calculate $P(h|D)$ provided we know the other three probabilities.

Bayes theorem:

$$P(h|D) = \frac{P(D|h)p(h)}{P(D)} \tag{1}$$

.

With the help of Bayes theorem v_{MAP} can be rewritten as

$$\begin{aligned} v_{MAP} &= arg \max_{v_j \in V} \frac{P(a_1, a_2, ..., a_n|v_j)P(v_j)}{P(a_1, a_2, ..., a_n)} \\ &= arg \max_{v_j \in v} P(a_1, a_2, ..., a_n|v_j)P(v_j) \end{aligned} \tag{2}$$

we can estimate the two terms in the above equation based on the training data. It is easy to estimate each of $P(v_j)$ simply by counting the frequency with which each target value v_j occurs in the training data. However, estimating the different $P(a_1, a_2, ..., a_n)$ terms in this fashion is not feasible unless we have a very large set of training data. The problem is that the number of possible instances times the number of possible target values. Therefore, we need to see every instance in the instance space many times in order to obtain reliable estimates.

$$v_{MAP} = arg \max_{v_j \in V} P(v_j|a_1, a_2, ..., a_n) \tag{3}$$

The naive Bayes classifier makes one simplifying assumption: the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the

conjunction $a_1, a_2, ..., a_n$ is just the product of the probabilities for the individual attributes: $P(a_1, a_2, ..., a_n|v_j) = \prod_i P(a_i|v_j)$. Substituting this in the above equation, we get the following equation.

Naive Bayes classifier:

$$v_{NB} = arg \max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j) \tag{4}$$

where v_{NB} denotes the target value output by the naive Bayes Classifier.

To summarize, the naive Bayes learning method involves a learning step in which the various $P(v_j)$ and $P(a_i|v_j)$ are estimated, based on their frequencies over training data. The set of these estimates corresponds to the learned hypothesis. This hypothesis is then used to classify the new instance by applying the equation of naive Bayes classifier.

3.4.2 Document Cleaning

As the documents collected are not directly amenable for learning we have to clean the documents. Currently, we only handle documents with the following content types: text/html, text/plain, Application/*, etc. - essentially documents whose content has text in it. The structure of a typical document is as follows: *meta data*, *HTTP Reply Headers*, *HTTP Reply Body*. We extract the plain text from the *HTTP Reply Body*. The information of interest is the URL and plain text. This plain text forms the content of our cleaned document. A cleaned document consists of the URL and the plain text extracted from *HTTP Reply Body*.

3.4.3 Initial Categorization for Learning

In a Bayes classifier we have to calculate $P(D|h)$. This means all the data used in learning must be pre-classified. Since the volume of training data is quite large, instead of classifying all documents manually, we have used the following technique to come up with class labels for the training documents. For each class or category we have identified keywords for the URL and for the content. The logic is if a keyword is

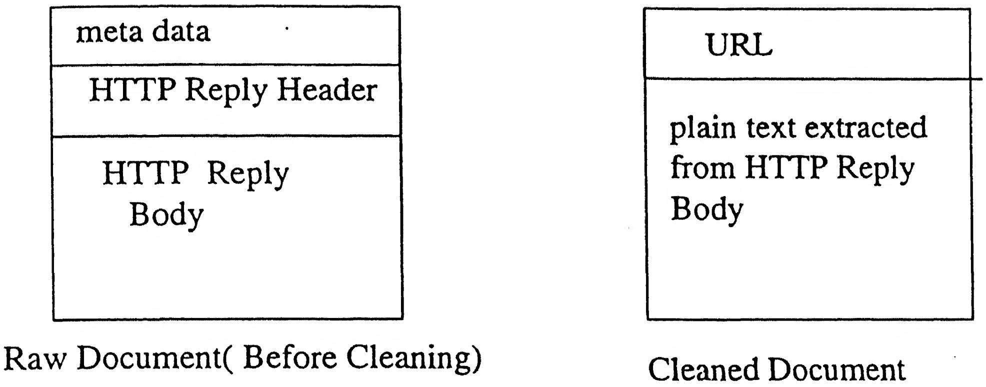

Figure 3.4: Document Structure

present in the URL or if a subset of keywords is present in the document the category of the document is identified as the category associated with the keyword(s).

Initial categorization is done using URL, docuement content and keywords. For verification purposes we have randomly selected some classified documents and verified it manually.

algorithm for Initial categorization

```
     For each category(i) find some key words which potentially
    represents that category.  For example, for category education,
    "edu", "research" etc forms the set of key words.
     For each document D
            if( keyword of category C, present in the URL)
                   category of document D is C.
            else if(maximum subset of keywords of
           category C belongs to the document D)
                   category of document D is C.
            else
                   category of document D is not known.
     verify categories of some randomly chosen documents.
```

What is meant by maximum subeset of keywords?. If 2 keywords of category

C_1 is present in the document D and 5 keywords of category C_2 is present in the document D, then document D is classified as category C_2 document.

3.4.4 Classification Algorithm

The next phase in Squid traffic analysis is classification. This is accomplished by the simple *naive Bayes classifier* [1] discussed above. The *naive Bayes classifier* performs well on many document classification problems and produces good results.

The cleaned documents along with their category are taken as training data for the naive Bayes classifier. For classifying other (that is documents not in the training set), this Bayes classifier is used to get the most likely category for the document. The words in a document are treated as the attributes for that document.

3.5 DelayPools

The delay pools concept, which is a bandwidth management scheme, has some drawbacks. The major one is that the *unused bandwidth of a Delay pool is wasted and cannot be utilized by other Delay pools.* Dynamic Delaypools [18] solve this problem and allow utilization of the full bandwidth.

The dynamic delay pools implementation is speedy and simple. The algorithm for bandwidth adjustment works in two stages.

- Allocation of Bandwidth to users per delaypool.
- Transfer of bandwidth from one pool to another.

The algorithm uses the *aggregate* number of tokens in the delay pool as the criterion for whether the user's link bandwidth is fully utilized or it skips the networks settings entirely. If number of tokens exceeds a cutoff value[presently 10% the aggregate delay pool size], indicating that there is some capacity being under-utilized, each individual user's bandwidth allocation is increased by a fraction. If the number of tokens is less than the cutoff value, indicating that the requests are backing up and link bandwidth is fully utilized, an individual user's bandwidth allocation is reduced by a fraction. This is shown in the flow chart below:

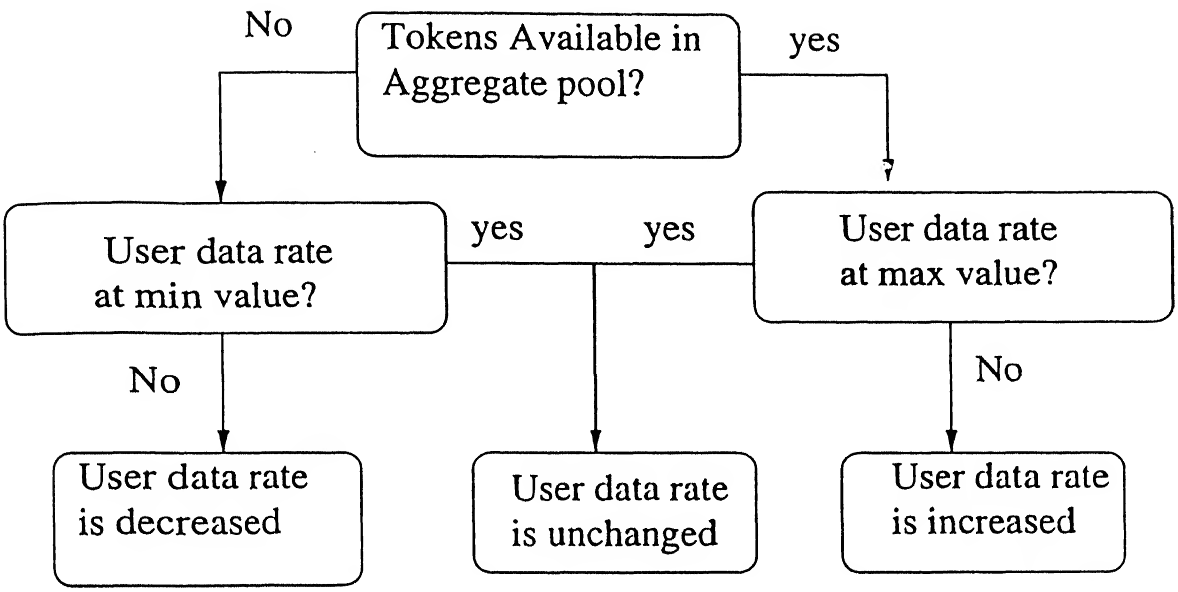

Figure 3.5: Transferring Bandwidth with in a Pool

Usually the link bandwidth is used both for the *restore* rate at which tokens are issued to the pool and maximum size of the pool. The aim of the algorithm is to keep the number of tokens in the pool as close to zero as possible. For practical purposes this value is usually set to 10% of the token restore rate and when the available tokens drop below this value users bandwidth is reduced. A value that is larger than zero is used, as it is difficult to precisely control the number of tokens in the pool.

Depending upon the number of tokens in the aggregate delay pool, the individual bandwidth a user receives can some times range between 100 bytes/sec and 10 kbps(it can be equal to the rate at which tokens are issued to the aggregate delay pool). The small minimum value is set to ensure that even at peak load users get some access, practically this limit is not usually reached. Setting the maximum value to the rate at which tokens are issued to aggregate pool ensures that even if there is only one user at that instant in the whole delay pool, that user can use all available resources.

Transferring bandwidth among delay pools:

Using a single delay pool allows bandwidth to be shared fairly among users. However, in many instances, we want bandwidth to be distributed unequally among different classes of users. This can be easily achieved by having several delay pools with different parameters. However, when several delay pools operate on the same server, some pools do not use all their allocated bandwidth while others are saturated, so the bandwidth which is not being utilized is allowed to be transferred to another delay pool as depicted in fig. 3.6.

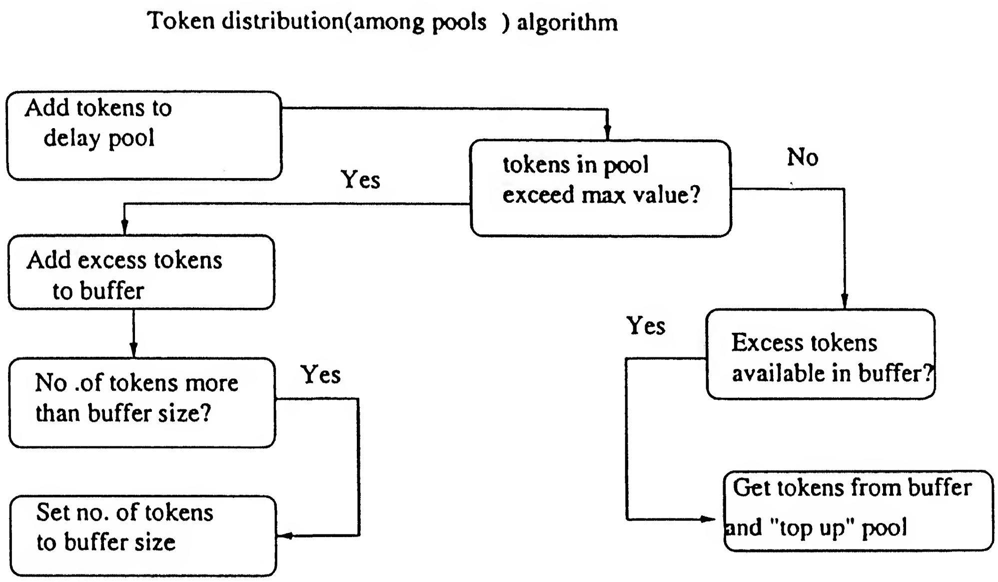

Figure 3.6: Transferring Bandwidth among Pools

The above algorithm checks the level of tokens in each delay pool after adding the token allocation available to that particular pool. If level of tokens is greater than the maximum allowed value, then excess tokens are transferred to a buffer and if a delay pool is encountered whose tokens are less than maximum permissible value, the tokens in the buffer are used to "top up" the pool to its maximum value or until the tokens in the buffer are exhausted. Accordingly, excess tokens that would be utilized by lightly loaded delay pools are distributed among the pools that are using their token allocation fully. The maximum size of the buffer is set to a few seconds worth of tokens.

3.5.1 Delay Priority

We propose a simple modification to the above dynamic delaypools and introduce the concept of *priority* among the delay pools. As the unused bandwidth of delay pools is distributed among other delay pools, the *priority* imposes an order on the distribution of unused bandwidth among the other delay pools. The delay pool with highest priority gets the unused bandwidth before the lesser priority delay pools.

The usage of such a priority is illustrated with a simple example. Suppose we have defined delay pools which accept different types of traffic based on the category say C_1, C_2, ...,C_n. If we want the delay pool associated with C_4 to get maximum bandwidth, then the priority setting of C_4 to a high value can solve the problem.

3.6 Policy Generation

The classification of traffic results in categories and a set of documents that belong to that category. As documents are related to the URLs, one can directly link the URLs with the categories.

(category, Set of Documents) $\rightarrow$ *(category, set of URLs)*, as each Doc(i) is related to URL(i).

The set of URLs along with the category forms the basis for the formation of policies. An administrator can regulate traffic by suitably changing the policies. In Educational Institutes, administrators want most bandwidth should be used for academic purposes rather than movies, music other unrelated stuff. So this category names, URL lists help in forming the policies.

3.6.1 Bandwidth

The traffic analysis helps in formulating bandwidth policies. Here we explain the steps for creating policies based on URL lists of category.

1. Design some named ACL elements based on a regular expression of URLs using ACL types of form *dstdomain_regex*, *dstdomain*, *url_regex*,*urlpath_regex*.

2. Design ACL lists like *http_access*, *delay_access* based on these named ACL

elements.

3. Define Delay Pool parameters based on the categories and define the Maximum Bandwidth and restore rate of each Delay Pool.

4. Use our *Delay Priority* concept to define the priority for each category, so that the unused bandwidth by that category is distributed according to the priority of other categories.

5. The ACL List *http_access* can be used to allow/deny the traffic.

Examples

`acl url_path_regex -i porn_url "porn_url_lists"` # assume porn_url_lists file contans the URLs found in the classification for category pornography.

`http_access deny por_url.` # deny traffic for this category.

These are simple policies to disallow porn content through the proxy. But one should find more reasonable access controls. We find the keywords with maximum frequency in the URL lists and form another acl element to complement the above.

`acl url_path_regex -i category_urlKeywords category_key_words` # the potential keywords in the URL lists for this category.

```
acl url_path_regex -i category_urlLists category_url_lists
http_access deny/access category_urlKeywords
http_access deny/access category_urlLists
```

But for some categories like `music` the keywords form the better acl, as the extension of music files can be easily found as keywords in the URL lists. To form policies based on the bandwidth along with the category, here we give an example for defining delaypools:

`delay_pools 4` # 4 delay pools are defined.

`delay_class 1 1` # delay pool 1 is of class type 1.

`delay_class 2 3` # delay pool 2 is of class type 3.

`delay_class 3 2` # delay pool 3 is of class type 2.

`delay_class 4 2` # delay pool 4 is of class type 2.

we explain the rest of parameters only for delay pool 1.

`delay_priority 1 3` # delay pool 1 has priority 3 . In distribution of unused bandwidth of other delay pools pool 1 gets the priority 3.

`delay parameters 1 aggregate_value` # aggregate_value is the amount of bandwith we want to allocate for delay pool 1.

Now define the traffic or requests which belong to delay pool 1. We can define many named acl elements and define the traffic that satisfies the acl elements as the delay pool 1 traffic.

`delay_access 1 category_urlKeywords category_urlLists` # the requests that have the keywords of category or that belong to the category url lists forms the traffic of delay pool 1.

3.6.2 Caching

The traffic analysis results also help in tuning the Squid to cache hot pages and deny caching of unwanted traffic. The ACL Lists like *no_cache* can be used along with the named ACL elements defined above to tune the caching.

For categories like *news* and *academic* the pages should be always cached and the pages of porn content should be denied.

Example:

`acl music_urlKeywords urlpath_regex -i music_urlKeywordslist.`

`no_cache DENY music_urlKeywords` # It denies the caching of music files.

Chapter 4

Implementation

4.1 Delay Priority

Delay Priority is the mechanism we have added to Dynamic Delaypools [18], to give more flexibility to the Squid Administrator in distributing link bandwidth. If bandwidth is unused by a certain pool, then its bandwidth is distributed among other pools according to specified priorities.

The user can specify the Delay Priority option through configuration parameters in squid.conf. To add this configuration parameter we have modified the Squid source code. The modified code mainly does the job of parsing the Delay priority configuration and setting its value for each of the pools, if defined.

To handle the updates of Delaypool bandwidth measures (tokens available, current bucket size, restore rate) an event called *DelayPoolsUpdate()* is defined in delaypools.c in Squid source code. In Dynamic Delaypools this has been modified to handle token distribution within a pool and in between the pools. We have modified the code for token distribution among the pools and implemented it in two phases *accumulation* and *distribution*. In accumulation phase if tokens of a pool cross a maximum permissible bucket size, then they are accumulated in a buffer. In *distribution* phase the tokens in the buffer are distributed to the pools sorted in order of priority, if the pool has capacity to hold the tokens.

Algorithm:

```
Delaypools Update()
{
        Accumulation phase:
                buffer ← collect the excessive tokens in each pool;
         Distribution phase:
                while( tokens in buffer available){
                        pool ← get next pool in sorted order of priority
                        if( token bucket of pool can still hold tokens)
                                distribute tokens in buffer to pool.
                }
}
```

The new tag DelayPriority syntax is as follows:

tag: DelayPriority <pool number> <priority>

usage: DelayPriority 2 5 # delaypool 2 has priority 5.

4.2 Squid Traffic Classification

We have implemented small components to achieve the goals of different phases of traffic classification. Here we explain the implementation of datacollector, HTML parser, and naive Bayesian classifier.

4.2.1 Data Collection

For collecting log files from Squid we just need to enable the store.log, access.log and cache.log files in squid.conf. Then Squid stores these log files in the location specified in squid.conf. For document collection we have implemented a simple external process, *datacollector*. Here we outline some implementation details.

The Squid proxy server stores the documents/pages which have matched cache preferences(high lifetime, refresh pattern, cache access controls) in the *cachefs* file

system. Whenever Squid stores the documents in the *cachefs*, it writes a record in *store.log* representing the *swapout* to the disk. The entries of *store.log* record namely *dirn, filenum* gives the location of document in the *cachefs*. Here we explain the logic of the *datacollector* in pseudo code.

```
DataCollector()
{
    while(Required number of Documents Collected) {
        Record ← get next recent record of store.log
        typeofDoc ← Record.field("type")
        sizeofDoc ← Record.field("sizes")
        if(the typeofDoc is understandable format like
              text/html, text/plain or other type and
              sizeofDoc is good enough for classification){
               fileloc ← getloc( cachefs location,
                            Record.field("dirn"),
                            Record.field("filenum"))
                copy the file to our local disk space
                    from fileloc
        }
    }
}
```

This is simple to implement and independent of Squid code. We have also implemented another way of data collection but it needs a small modification to Squid. For the sake of completeness we mention it here. It is basically an IPC mechanism, where Squid proxy sends an IPC message with the information of *type, size, dirn and filenum* to our external process. Then our external process makes use of this information to locate the document and copy it to the local folder/disk space.

4.2.2 Data Cleaning

The documents collected from Squid *cachefs* have the following elements: metadata, HTTP Reply header, HTTP Reply Body.

- *metadata reading:*

  When Squid stores objects to disk, it first writes the meta data[19].This is done with a couple of functions, *storeSwapMetaBuild()* and *storeSwapMetaPack()* from storeswapmeta.c in the Squid source code. So when reading the swap object/file from *cachefs* reverse of the above process has to be done. We call *storeSwapMetaUnpack()* to read the meta data.

- *HTTP Reply Header reading:*

  The swap file/object contains the HTTP Status Line. We read status line and then HTTP header. The structure of HTTP Header in the swap file is as follows:

  ```
  message-header = field-name ":" [ field-value ] CRLF
  field-name     = token
    field-value    = *( field-content | LWS )

    where CRLF is \\r\\n and LWS is white spaces.
  ```

  Using the above structure we read the HTTP Reply header from the swap file/object.

- *HTTP Reply Body reading:*

  HTTP Reply Body contains the actual content of the document. The second component URL can be retrieved from meta data. The <URL, content> forms the structure of the document. As we are collecting the documents of type text/html, text/plain, Application/* to collect content from text/html type of documents we need a *HTML parser*. With the help of *HTML parser* we have collected the content from swap files/objects.

Once we have cleaned the documents, we do initial categorization on these documents.

4.2.3 Naive Bayes Classifier

Here we briefly explain the algorithm for the naive Bayes classifier [1].

LEARN_NAIVE_BAYES_TEXT(*Examples*, *V*)

description: Examples are a set of documents along with their target values, V is the set of possible target values. This function learns the probability terms $P(w_k|v_j)$ describing the probability that a randomly drawn word from a document in class v_j will be English word w_k. It also learns the prior probabilities of the class. Normally, each class is assumed equally likely.

steps:

1. *Collect all words, punctuation, and other tokens occurring in Examples.*

Vocabulary $\leftarrow$ the set of all distinct words and other tokens occurring in any text document from *Examples*.

2. *Calculate the required* $P(v_j)$ *and* $P(w_k|v_j)$ *probability terms*

For each target value v_j in V do {

$docs_j \leftarrow$ the subset of documents from *Examples* for which the target value is v_j

$P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$

$Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$

$n \leftarrow$ total number of distinct word positions in $Text_j$

for each word w_k in *Vocabulary* {

$n_k \leftarrow$ number of times word w_k occurs in $Text_j$

$P(w_k|v_j) \leftarrow \frac{|n_k|+1}{n+|Vocabulary|}$

}

}

CLASSIFY_NAIVE_BAYES_TEXT(*Doc*)

description *Return the estimated target value for documents Doc. a_i denotes the world found in the ith position with in the Doc.*

- *positions* $\leftarrow$ all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return v_{NB} where

$$v_{NB} = arg \max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j) \tag{1}$$

In the above algorithm *Examples* means training documents and *class* means category of document. In this algorithm the position of words in the document is taken as independent.

While implementing the algorithm we have encapsulated all the words in the same class documents in a *hash table* along with the frequency. We have also maintained *count* of total words in that hash table. Then the calculation of $P(w_k|v_j)$ becomes simple that n_k, n , $|vocabulary|$ are available straight away.

Using the Naive Bayesian Classifier the documents have been classified. But our interest is to form policies. As each document related to a URL, we can form URL lists of each category. These URL lists help in forming the policies restricting sites and setting priority, bandwidth etc.

We have chosen the number of training documents ($N_{training}$) for each category in range of 100-250 initially. Then we increased the size of training data till we achieve stable recognition rates. The actual details are presented in chapter 6.

Chapter 5

HTTP Tunnelling - A study

5.1 HTTP Tunnelling

Squid allows setting of policies for restricting user access based on many configuration parameters like source, destination, domains, time, regular expression on URLs etc. Squid allows HTTP tunnelling to support SSL and TLS[3]. This forms a loophole and policies can be bypassed. Generally, users use tunnelling tools to over come access controls. So there is a need to find a mechanism for enforcing policy strictly. We have studied the concept of HTTP Tunnelling and provided possible solutions to enforce policy strictly. The usage of such tunnelling tools can be broadly divided into two categories,

- Increase speed: To speed up the download of certain software or 'good' content which is acceptable to current policy (e.g. academic content)
- Violate current restrictions: To download data/pages which are not permitted by current policy (e.g. pornography).

In the first case tunnelling can be permitted. However, in the second case it should be prohibited. Here we briefly discuss the usage of such tunnelling tools and possible solutions to stop the tools from tunnelling.

5.1.1 HTTP Tunnelling Clients

HTTP tunnelling clients connect to outside servers of their own or some free servers through the proxy server using the HTTP CONNECT method. Requests of method CONNECT are not interpreted by the proxy server, they are just forwarded to the server:port. Thus a tunnelling client connects to a server, further communication

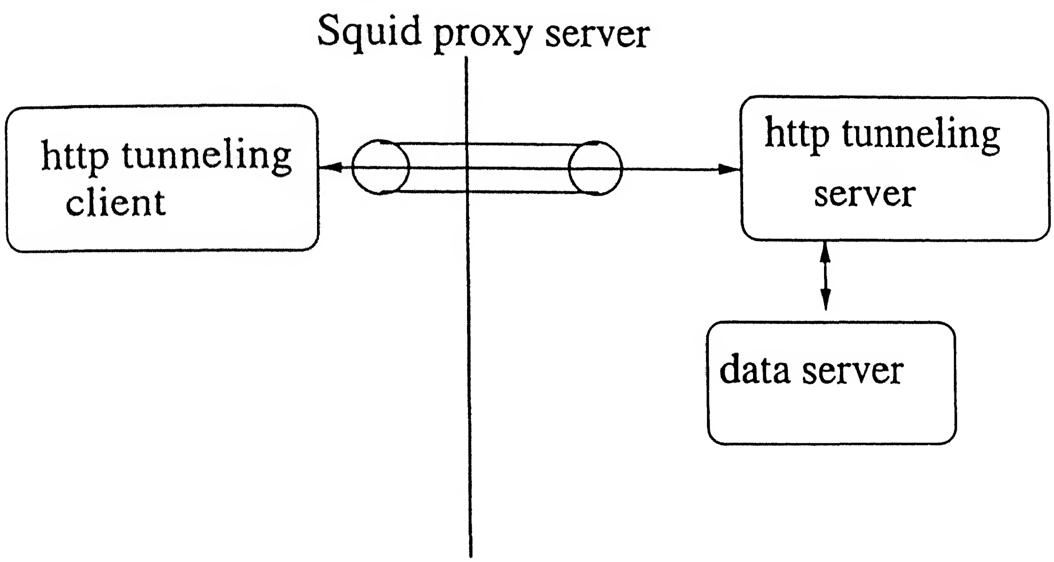

Figure 5.1: Working Mechanism of Http-tunneling clients

gives the links/pages for downloading, now the tunnelling server connects to the actual target server gets the data and forwards it back to the client through the proxy server. The data exchanged between the target server and client is simply forwarded in both directions by the proxy server, which just works as a tunnel between the client and the target server.

There are many tunnelling-clients and tunnelling-servers [11][10]. The services range from free to paid.

5.1.2 Socks2http

Socks2HTTP software components work like an agent converting SOCKS v.x requests into HTTP requests and tunnelling them through an HTTP proxy. It works

as a lightweight SOCKS proxy, tunnelling TCP/UDP traffic through a HTTP proxy or firewall. SOCKS allows programs to traverse firewalls on any port number and is used by many popular programs, like Morpheus, MSN Messenger, ICQ, AOL IM, FTP, Telnet and many others. Many companies restrict firewall traversals only to HTTP requests, disabling SOCKS proxy. Socks2HTTP provides a lightweight SOCKS server for the SOCKS client, performing its connection through an HTTP proxy. Socks2HTTP will try to connect to the target server directly, if it fails it will then use a gateway. For example, TotalRC.net provides a public gateway (currently www.totalrc.net, port 80) but due to high load it can be slow at times.

5.1.3 An Approach

Squid proxy server supports HTTPS, SSL and TLS protocols, which can tunnel. HTTP CONNECT method can be used to connect to proxy on different ports. This allows a tunnelling-client to use one of the ports to CONNECT to the Squid proxy. If we restrict the CONNECT method to only SSL ports then we can partially stop such tunnelling clients. However, some tunnelling clients use the SSL ports to connect to tunnel servers even for ordinary data and these can still tunnel through the proxy server.

Squid ACL mechanism allows definition of such restrictions. A policy of the following kind implements such a restriction.

- acl CONNECT method CONNECT
- acl SSL_ports port 443 563
- http_access deny CONNECT !SSL_ports #deny connect other than SSL ports.

The problem with the above solution is that if users are trying to use tunnelling clients for speed up (but does not violate access policy), we are potentially restricting the traffic on such ports.

We can provide less aggressive solutions that deny CONNECT only on the unsafe ports. But this way we are potentially opening up the ports for CONNECT and allowing tunnelling clients.

- acl SSL_ports port 443 563
- acl Safe_ports port 80 # http
- acl Safe_ports port 21 # ftp
- acl Safe_ports port 443 563 # https, snews
- acl Safe_ports port 70 # gopher
- acl Safe_ports port 210 # wais
- acl Safe_ports port 1025-65535 # unregistered ports
- acl Safe_ports port 280 # http-mgmt
- acl Safe_ports port 488 # gss-http
- acl Safe_ports port 591 # filemaker
- acl Safe_ports port 777 # multilink http
- http_access deny !Safe_ports # Deny requests to unknown ports
- http_access deny CONNECT !Safe_ports # Deny CONNECT to other than Safe ports

Here we allow the users to use tunnelling-clients, but we can block the tunnelling servers, if traffic crosses certain limits. Identification of such tunnelling- servers can be done from the access.log fields namely server and bandwidth. If bandwidth serviced by a certain server crosses a limit, then add that tunnel-server to the blocked list. In this way we can block some tunnelling-servers. But this should be a continuous process to handle new tunnelling-servers.

However, free servers (they are available in large number) can be used as tunnelling servers. To over come these **iptables, ipchains** rules and *CBQ protocols* can be used.

If we want to open the CONNECT service on safe ports and want to restrict the bandwidth, we can define a Delay pool based on CONNECT traffic.

Chapter 6

Testing & Results

A machine learning problem generally has the three parts: namely source of experience E, class of tasks T and the performance measure P.

- source of experience E : This helps in training. Any training data that helps to improve the performance of the tasks is said to be a source of experience. These form the training examples. In our problem the log data and documents collected form the training examples.

- Tasks T : A set of tasks that learn from training examples and try to improve performance P after training.

  In our problem *Squid traffic classification* is the task.

- Performance measure P: The selection of performance generally depends on the problem, its understanding, and variables to be optimized. Here we selected the *percentage of bandwidth consumed for each of the categories.* Our task is to improve the percentage of bandwidth for certain classes and reduce the bandwidth available to other classes. More formally $(category, bandwidthusage)$ tuples form the measure. However, note that in our case this breaks down into two independent measures since categorization of URLs is handled by the naive Bayes classifier and bandwidth utilization is managed independently by priority based dynamic delay pools.

So first we classify the traffic, then formulate policies based on the classification results and finally set these policies in the Squid proxy. Performance is measured independently for classification and for bandwidth management.

Experiments:
Internet data has been categorized into 9 categories.

- news
- sports
- entertainment
- music
- science technology
- chat
- movie
- academic
- porn

We have taken certain keywords for each category to identify the category of the document in initial categorization. Here we mention some of these keywords.

- news: news, vaartha, timesofindia, cnn, eenadu, hindu
- sports: sport, cricket, football, tennis, chess, athletic .
- entertainment: greetings, dance, hero , celebrity, marriage, jokes
- music: music, song, audio, mp3, ram.
- science & technology: Science, technology, breakthrough. For this category, we have collected manually some documents and added to this category for learning.

- chat: chat, msg, messenger, notify, pager.
- movie: movie, video, dat, asf, asx, mplayer, realplayer.
- academic: research, .pdf, .ps, .gz,.bz, download , package, edu ,university ,exam , academic.
- pornogrpahy: nude, porn, sex, adult, xxx, lesbian, blowjob, fuck, naked.

The classification of traffic results in URL lists for each category. A simple method is to use the http_access, url_regex_path to design policy.

`acl url_path_regex porn_url porn_url_lists` # *porn_url_lists contains the URLs found in the classification.*
`http_access deny por_url.` # *deny traffic for this category*

Above rules provide simple policies to deny some kinds of porn content through the proxy. But one should find more reasonable access controls. We find the keywords with maximum frequency in the URL lists and form another acl element to complement the above.

`acl url_path_regex -i category_urlKeywords category_key_words` # *the potential keywords in the URL lists for these category.*
`acl url_path_regex -i category_urlLists category_url_lists`
`http_access deny/access category_urlKeywords`
`http_access deny/access category_urlLists`

But for some categories like `music` the keywords give better control, for example extensions of music files can be easily found as keywords in the URL lists.

To form policies based on the bandwidth along with category, here are some examples for defining the delaypools.

`delay_pools 4` # *4 delay pools have defined*
`delay_class 1 1` # *delay pool 1 is of class type 2* `delay_class 2 3` # *delay pool 2 is of class type 3*

`delay_class 3 2` *# delay pool 3 is of class type 1*

`delay_class 4 2` *# delay pool 4 is of class type 2*

we explain all parameters only for delaypool 1.

`delay_priority 1 3` *# delay pool 1 has priority 3. In distribution of unused bandwidth of other delay pools pool 1 gets the priority 3.*

`delay parameters 1 -1/-1 8000/8000` *# strictly limit each host to 64kbps(plus overheads), with no overall limit*

Now define the traffic or requests, which belong to delay pool 1. We can define many named acl elements and define the traffic that satisfies the acl elements as the delay pool1 traffic.

`delay_access 1 category_urlKeywords category_urlLists` *# the requests that belongs to keywords of category or that belongs to category url lists forms the traffic of delay pool 1.*

6.1 Results

We set policies based on the URL lists for categories as explained earlier. We collected the log files and documents after setting the policies. Here we present some of results in terms of our performance measure for a day by taking average of a period of 7days.

6.1.1 Deny Pornography Content

We have set the policies to deny pornographic content using keywords and list of URLs. We have compared bandwidth usage and number of files downloaded that belong to this category before and after setting the policies.

`Using Access.log:`

Here we present statistics (*average values per day*)for a period of a week, before and after setting policies. Downloading a page like www.google.com results in sending several requests for text, for all images in that page, banners etc. A file download corresponds to a single file, so for google it will result in multiple downloads.

We have classified the URL entries in the access.log as follows:

Content	files downloaded			bandwidth(KBytes)		
	before	after	change	before	after	change
porncontent	184204	48737	-73.54%	981596	1,86905	80.9%
other documents	919923	979920	6%	11082749	12192349	1.5%

Table 6.1: Performance measure(two categories)

Content	files downloaded		
	before	after	change
porncontent	4126	837	-79.7%
other documents	25874	29163	12.7%

Table 6.2: Performance measure(documents)

- The frequent keywords in URL lists(output of naive Bayes classifier)for pornography.
- Keywords used in the initial categorization.

`Using the Cached Documents:`
We have collected 30,000 documents for a period of a day before and after setting policies. We have applied the naive Bayesian classifier on these documents. The characteristic of the documents is that

- the content of the file is understandable like text/html or text/plain
- the size of the document is at least 8Kbytes(before cleaning).

So the bandwidth here will not signify much, as we are not considering the images, song files like .mp3, .rm files etc.

all categories: Here we classified the data collected from the access.log file into all nine categories.

Category	files downloaded			bandwidth		
	before	after	change	before	after	change
news	121888	189773	55.5%	15,45,548	1856768	20.1%
academic	245656	235877	-3.08%	28,87,556	27,67,778	-4.14%
chat	211839	200754	-5.2%	70,566	72,567	2.83%
music	36785	25785	-29%	10,45,334	856777	18.03%
movie	7622	10476	37.4%	3,55,765	378992	6.03%
sport	103023	230445	123.68%	13,53,467	15,67844	15.86%
entertainment	36056	31546	-12.50%	5,77,694	7,46654	29.32%
science technology	7554	8973	18.74%	1,07,772	1,87556	74.03%
pornography	184204	48737	-78.76%	9,81,596	1,86,905	80.9%
unclassifed	21446	18335	-	85,798	96552	-

Table 6.3: Bandwidth distribution (All categories)

The high change in sports category is due to frequent visits to cricket and tennis. The cache hit rate for sport category is high.

6.1.2 Bayesian Learning Classification:

Initially we collected 30,000 cleaned documents. We trained on 5000 documents using initial categorization. For each category we used around 200 - 400 documents in training. For pornography, chat and academic categories we used around 800 -1000 documents for training. Then we classified the 25000 documents using the Bayes classifier and found the frequency of URL along with documents. We added the different documents of the same URL which belongs to the same category to trained data. This way exactly 1000 training documents were obtained for each category.

We collected 20,000 more documents and classified them using the naive Bayesian classifier. The accuracy of classification for the categories is as follows:

The verification of results done with the keywords and manually. When the

number of domains is less we have done the verification manually. For the category like *academic*, the presence of the keyword **.edu** is good enough.

- chat - 97% . Verification of this category is done with keywords in URL.
- pornography — 98%. The total number of domains we got around 150 and these sites are frequently accessed. But some home pages in geocites and yahoo groups also found.
- academic: 98%. The verification of this category is also done with the keywords in URL and documents.

For other categories the accuracy was as follows:

- music : 92%. In the remaining 8% documents, the naive Bayes classifier identifying the category as music, but the probability of V_{MAP} is not greater than 0.5 value.
- movie : 88.4%.
- entertainment : 86%.
- news : 94%. The number of vocabulary words for this category is high.

6.1.3 HTTP Tunnelling Policies and Results

In this section we show the result of setting policies to inhibit tunnelling. The results are in the form of entries in access.log and not in terms of percentages since it is not possible to measure such percentages.

After setting the policy as **http_access deny !SSL_ports**, Squid denies any requests to HTTP CONNECT from internal clients. Here we give some of the denied entries in the *access.log*.

Entries in access.log

1042356698.348 2 172.31.4.166 TCP_DENIED/403 1004 CONNECT login.oscar.aol.com:5190 - NONE/- -
1042358767.840 2 172.31.97.66 TCP_DENIED/403 979 CONNECT

www.aim.com:80 - NONE/- -
1042359662.590 2 172.31.81.101 TCP_DENIED/403 987 CONNECT
scs.yahoo.com:5050 - NONE/- -
1042371314.932 3 172.31.4.185 TCP_DENIED/403 1008 CONNECT
messenger.hotmail.com:1863 - NONE/- -
1042371375.310 2 172.31.4.185 TCP_DENIED/403 1008 CONNECT
messenger.hotmail.com:1863 - NONE/- -

comment: In all log entries the entries indicate that clients (172.31..*) make requests to connect to some server:port(e.g. in second record scs.yahoo.com:5050), our policy implementation denies such requests*

1042374622.617 2 172.31.89.251 TCP_DENIED/403 987 CONNECT
24.158.24.189:3074 - NONE/- -
1042374622.631 2 172.31.89.251 TCP_DENIED/403 983 CONNECT
24.216.52.6:2829 - NONE/- -
1042374622.660 2 172.31.89.251 TCP_DENIED/403 987 CONNECT
24.44.250.169:3627 - NONE/- -
1042374622.681 3 172.31.89.251 TCP_DENIED/403 985 CONNECT
24.188.59.72:1407 - NONE/- -
1042374622.705 2 172.31.89.251 TCP_DENIED/403 985 CONNECT
24.186.225.6:2428 - NONE/- -
1042374622.727 2 172.31.89.251 TCP_DENIED/403 989 CONNECT
24.190.212.108:2941 - NONE/- -
1042374622.749 2 172.31.89.251 TCP_DENIED/403 989 CONNECT
24.190.162.180:3198 - NONE/- -
1042374622.770 2 172.31.89.251 TCP_DENIED/403 985 CONNECT
24.184.12.52:1397 - NONE/- -
1042374622.792 2 172.31.89.251 TCP_DENIED/403 989 CONNECT
24.205.101.140:2423 - NONE/- -
1042374622.815 2 172.31.89.251 TCP_DENIED/403 989 CONNECT
24.184.116.176:1837 - NONE/- -

1042438449.786 1 172.31.84.32 TCP_DENIED/403 1012 CONNECT 205.188.246.217:80 - NONE/- -

In the above entries also the requests for server:port number with HTTP CONNECT method is denied. This is the effect of setting the above policy.

We had run some HTTP tunnelling clients and collected their error messages, when they failed to connect to Squid for tunnelling.

direct get:

As the HTTP GET is allowed by Squid proxy, some clients use the get method with different threads and speedup the downloading. Using direct get option the tools successfully download data as they use only "GET" method.

http tunnelling (GET + CONNECT):

when we block the CONNECT on SSL ports , the tunnelling tools cannot use CONNECT on these ports. Here we are giving log of such a tool, which fails to tunnel.

ernet-connect: Tue Feb 25 13:17:06 2003 Connected.
Tue Feb 25 13:17:06 2003 CONNECT www.ai.mit.edu:80 HTTP/1.0
Tue Feb 25 13:17:06 2003 Proxy-Authorization: Basic c3VkaGFrYXI6bmV3cHV6emxl
Tue Feb 25 13:17:10 2003 HTTP/1.0 403 Forbidden
Tue Feb 25 13:17:10 2003 Server: Squid/2.4.STABLE7
Tue Feb 25 13:17:10 2003 Error occured!
Tue Feb 25 13:17:10 2003 Wait 5 second for retry
Tue Feb 25 13:17:15 2003 Connecting proxy 202.141.40.24 [IP=202.141.40.24:3128]
Tue Feb 25 13:17:16 2003 Connected.
Tue Feb 25 13:17:16 2003 CONNECT www.ai.mit.edu:80 HTTP/1.0
Tue Feb 25 13:17:16 2003 Proxy-Authorization: Basic c3VkaGFrYXI6bmV3cHV6emxl
Tue Feb 25 13:17:16 2003 HTTP/1.0 403 Forbidden
Tue Feb 25 13:17:16 2003 Server: Squid/2.4.STABLE7
Tue Feb 25 13:17:16 2003 Error occured!
Tue Feb 25 13:17:16 2003 Wait 5 second for retry
Tue Feb 25 13:17:21 2003 Connecting proxy 202.141.40.24 [IP=202.141.40.24:3128]

Tue Feb 25 13:17:24 2003 Connected. `# connection to ernet proxy is made`
Tue Feb 25 13:17:24 2003 CONNECT www.ai.mit.edu:80 HTTP/1.0 `trying to make CONNECT request`
Tue Feb 25 13:17:24 2003 Proxy-Authorization: Basic c3VkaGFrYXI6bmV3cHV6emxl
Tue Feb 25 13:17:24 2003 HTTP/1.0 403 Forbidden `# ernet proxy denies that request`
Tue Feb 25 13:17:24 2003 Server: Squid/2.4.STABLE7 `# denial error came from ernet proxy`
Tue Feb 25 13:17:24 2003 Error occured!
`#this kind of message repeated in the log many times`
. Tue Feb 25 13:19:13 2003 Wait 5 second for retry
Tue Feb 25 13:19:37 2003 Encountered maximum error. This part job is halted.
Tue Feb 25 13:19:37 2003 Didn't receive any data within 99 tries, should stop.

http2sock Here we are providing log of the software, which uses socks server on its machine and tries to use proxy to connect to a server. This is a log of a client using a socks server to download system_performability.pdf a link from Internet explorer resulted in Google search.
Tue Feb 25 14:29:30 2003 Connecting proxy localhost [IP=127.0.0.1:1080]
Tue Feb 25 14:29:30 2003 Connected.
Tue Feb 25 14:29:30 2003 Socks: 05 01
Tue Feb 25 14:29:30 2003 Socks: 05 00
Tue Feb 25 14:29:30 2003 Socks: 05 01
Tue Feb 25 14:29:30 2003 Socks: 05 00
Tue Feb 25 14:29:30 2003 GET /cs340-22/docs/materials/system_performability.pdf HTTP/1.1
Tue Feb 25 14:29:30 2003 Host: cs-tl3.cs.virginia.edu
Tue Feb 25 14:29:30 2003 Accept: */*
Tue Feb 25 14:29:30 2003 Referer: http://www.google.com/search?hl=enlr=ie=ISO-8859-1q=%22robot+movement%22+%2B+%22pdf%22&btnG=Google+Search

Tue Feb 25 14:29:30 2003 User-Agent: Mozilla/4.0 (compatible; MSIE 5.00; Windows 98)

Tue Feb 25 14:29:30 2003 Pragma: no-cache

Tue Feb 25 14:29:30 2003 Cache-Control: no-cache

Tue Feb 25 14:29:30 2003 Connection: close

Tue Feb 25 14:30:25 2003 HTTP/1.1 404 Not Found

Tue Feb 25 14:30:25 2003 Date: Tue, 25 Feb 2003 09:02:04 GMT

Tue Feb 25 14:30:25 2003 Server: Apache/1.3.27 (Unix) PHP/4.3.0 mod_ssl/2.8.12 OpenSSL/0.9.6g

Tue Feb 25 14:30:25 2003 Connection: close

Tue Feb 25 14:30:25 2003 Transfer-Encoding: chunked

Tue Feb 25 14:30:25 2003 Content-Type: text/html; charset=iso-8859-1

Tue Feb 25 14:30:25 2003 Error occured!

Chapter 7

Conclusions and Future Work

In this thesis we have proposed methods to better manage access and bandwidth to an internet link through a proxy server. Access control is done by categorizing URLs into different static categories using a naive Bayes classifier. Bandwidth control is done using priority based dynamic delay pools. We also tested some methods to inhibit HTTP tunnelling.

The initial results are encouraging. For example pornographic content reduced significantly and the bandwidth was well utilized for other categories. Policies using priority based dynamic delay were implemented and bandwidth utilization improved. Log traces show that the methods to inhibit HTTP tunnelling they do manage to stop tunnelling requests.

7.1 Future Work

Currently the various methods are not tied together to provide a policy making environment which operates dynamically and automatically. One could partially automate the policy implementation using the syntax for specifying policy as defined in squid.conf.

We have found and experimented with policies for bandwidth management and restricting traffic in certain categories. We also have some partial policies for tuning the cache. There is considerable scope for tuning the cache using either machine

learning or other techniques. Squid provides many mechanisms to set policy. We have not explored all such mechanisms and there is a lot of scope for finding other policies using machine learning techniques. However, care is needed in identifying the task, source of experience and the performance measure suitable for such policies.

We have not considered tunnelling traffic in terms of documents. It would be interesting to find a way to analyze this traffic. But the main hurdle will be the format of tunnel data as in most cases it is encrypted. For collecting such traffic one can use packet filters and collect data outside the squid environment.

So far dynamic delay pools offer bandwidth management in Squid Proxy. But it should be interesting to come up with an altogether new approach. Dynamic delay pools is based on the token bucket algorithm, but there are other QOS algorithms that can be tried for bandwidth management.

Appendix A

Announcement

The `ernetproxy.iitk.ac.in(202.141.40.24)` is another proxy server for internet. One can use this server to download all kinds of data and files. There is no explicit limitation on the type of files. The log data collected from the user requests,responces is used for research purposes. Mainly we are using the log data

- To find the patterns of requests(w.r.t time of day, total requests by type etc.).
- To find the policies for the bandwidth sharing.
- To find the prefetching, sharing policies,
- To find a reasonable authentification policies,
- To identify what are the possible ways to overcame the proxy restrictions

THE PRIVACY OF EACH USER IS PROTECTED.

To use these ernetproxy,simply set

- proxy : ernetproxy.iitk.ac.in
- portno : 3128.

Appendix B

Delay Priority

Here we give the newly added configuration parameter to squid.conf file.

#TAG: delay_priority
This is used to assign the priority for each of Delay pools.
If b/w of one Delay pool is not used then it is assigned to the
other delay pools according to the priority.
#
#Example:
delay_priority 1 3 #pool 1 has priority 3
delay_priority 2 1 #pool 2 has priority 1
delay_priority 3 2 #pool 3 has priority 2
#Default:
none

Bibliography

[1] Tom M Mitchell. *Machine Learning*, McGraw-Hill International Edition, 1997.

[2] S. Tanenbaum., *Computer Networks*. Printice-Hall of India, Pvt. Ltd.,3^{rd} Edition, pages 381-384, 1999.

[3] Tunneling TCP based protocols through web proxy servers. *www.web-cache.com/Writings/Internet-Drafts/draft-luotonen-web-proxy-tunneling-01.txt*

[4] Squid proxy server. URL *www.squid-cache.org*

[5] Squid Frequently Asked Questions *www.squid-cache.org/Doc/FAQ/*

[6] Squid configuration file: squid.conf.

[7] Squid configuration Manual. *www.visolve.com/squid24s1/contents.html*

[8] *david@luyer.net*

[9] Upgrading to TLS Within HTTP/1.1 *www.ietf.org/rfc2817.txt*

[10] HTTP-Tunnel Corporation – Networking Products For Corporate Communications. *www.http-tunnel.com*

[11] Cutting Edge Web Applications. *www.totalrc.com*

[12] Squid Cache Logfile Analysis Scripts. *www.squid-cache.org/Scripts*

[13] Calamaris: Log analyzer *http://cord.de/tools/calamaris/*

[14] Webalizer: log analyzer *http://mrunix.net/webalizer*

[15] Cache Digest Specification - Version 5. *www.squid-cache.org/CacheDigest/cache-digest-v5.txt*

[16] Hyper Text Transfer Protocol – HTTP/1.1
www.ietf.org/rfc/rfc2616.txt

[17] A. Rousskov and V. Soloviev. *A Performance Study of the Squid Proxy on HTTP/1.0.* In *World Wide Web*, pages 47-67, June 1999.

[18] Chamara Gunaratne, Gihan Dias (University of Moratuwa) *Using Dynamic Delay Pools for Bandwidth Management* URL: *www.2002.iwcw.org/*

[19] Squid programmers guide *www.squid-cache.org/Prog-Guide/prog-guide.html*

[20] Lang, K. *Newsweeder: Learning to filter netnews.* In Priedits and Russel(Eds.), *Proceedings of 12th International Conference on Machine Learning (pp. 331-339).* San Francisco: Morgann Kaufmann Publishers.,1995.

[21] Rish, I. *An empirical study of the naive Bayes classifier.* in IJCAI-01 workshop on "Empirical Methods in AI"., 2001.

[22] Rish, I., Hellerstein, J., and Jayram, T.S. *An analysis of data characteristics that affect naive Bayes performance.* IBM Technical Report RC21993, 2001.

[23] I. Rish. *Advances in Bayesian Learning*, a short tutorial presented at IC-AI'2000, Las Vegas, June 2000.